UNITED STATES PATENT APPLICATION FOR

**TWO TIER ARRANGEMENT FOR THREADS SUPPORT
IN A VIRTUAL MACHINE**

Inventors:
Venky Krishnan
Geethan Manjunath
Devaraj Das

BACKGROUND OF THE INVENTION

Field of Invention

The present invention pertains to the field of
software systems. More particularly, this invention
relates to a two tier arrangement for threads support
in a virtual machine.

Art Background

Computer systems and devices having embedded
processing resources are typically implemented in a
variety of differing architectures. Each
architecture is usually defined by a particular
instruction set, hardware register set, and memory
arrangement, etc. In addition, such computer systems
and devices typically include an operating system
which is adapted to its particular architecture. The
operating system and architecture provide a platform
for execution of software tasks.

Typically, software tasks such as application
programs which are written or compiled to be executed
on a particular platform may be referred to as native
code. A software task in the native code of a
particular platform usually does not run on other non
compatible platforms.

A virtual machines is a software environment
that enables application programs to execute on a
variety of differing platforms. The application
programs which execute under such a virtual machine
usually take the form of a stream of instructions
each of which conforms to a predefined instruction

set supported by the virtual machine. A virtual
machine implemented on a particular platform
typically interprets each of the instructions in the
stream and provides emulation of the instructions in
5   the native code of the particular platform.   In
addition, a virtual machine may include threads
support. Threads support may be defined as
functionality that enables the parallel execution of
multiple software tasks each of which is referred to
10   as a thread.

One example of a virtual machine that enables
application programs to execute on a variety of
differing platforms and that provides threads support
15   is a Java virtual machine. A typical Java virtual
machine functions as an interpreter for Java
application programs. A Java application program
typically take the form of a stream of Java byte code
instructions and the Java virtual machine emulates
20   each Java byte code instruction using the native code
of the particular platform under which the Java
virtual machine executes. A typical Java virtual
machine provides threads support which is defined by
the java.lang.Thread class.

25

A typical virtual machine invokes various
services provided by the operating system of the
underlying platform including services that provide
threads support. Typically, the nature and extent of
30   such operating system services varies among operating
systems. For example, some operating systems provide
extensive threads support services while other
operating systems provide little or no threads

support services. In addition, the details of any
threads support services usually varies widely among
operating systems. Such variation in operating
system services usually complicates the process of

5   adapting a virtual machine to different platforms.
Unfortunately, such complications usually increase
the development time and cost of implementing virtual
machines on different platform. Moreover, such
complications usually increase the cost of software

10   support for virtual machines on different platforms.

SUMMARY OF THE INVENTION

A software system is disclosed with a two tier arrangement for multi threading in a virtual machine that enhances the adaptability of the virtual machine to different platforms. The two tier arrangement includes a threads interface layer in the virtual machine and an underlying native threads interface layer. The threads interface layer includes a set of methods that provide thread support in the virtual machine according to a standard threads interface associated with the virtual machine. The native threads interface layer includes a set of methods that adapt the methods of the threads interface layer to a platform which underlies the software system.

The threads interface layer provides a stable interface for application programs and other tasks that benefit from thread support in the virtual machine. The interactions between the threads interface layer and the native threads interface layer are arranged to shield the virtual machine from the particulars of the underlying operating system, thereby enhancing portability of the virtual machine and ease of adapting to upgrades in the underlying operating system.

Other features and advantages of the present invention will be apparent from the detailed description that follows.

Attorney Docket No.: 10981455

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is described with respect to particular exemplary embodiments thereof and reference is accordingly made to the drawings in which:

Figure 1 shows a software system that includes a two tier arrangement for providing thread support in a virtual machine;

Figure 2 is a state diagram showing the valid states of a thread in one embodiment;

Figure 3 shows a method for providing threads support for a virtual machine in a software system.

DETAILED DESCRIPTION

Figure 1 shows a software system 10 that includes a two tier arrangement for providing thread support in a virtual machine 12 according to the present teachings. The software system 10 may be implemented on wide variety of hardware platforms including computer systems and devices having embedded processing resources.

The virtual machine 12 provides a software environment that enables execution of application programs such as an application program 20 on a wide variety of differing platforms. The virtual machine 12 provides a set of standardized programming interfaces to the application program 20 including standardized interfaces for multi-thread support.

In one embodiment, the virtual machine 12 is a Java virtual machine and the application program 20 is a java application. The following description focuses on an example embodiment in which the virtual machine 12 is a Java virtual machine. Nevertheless, the two tier arrangement for threads support in the software system 10 is readily adaptable to other types of virtual machines.

The software system 10 includes an operating system 18. The operating system 18 together with a particular hardware architecture for which it is implemented provides a platform for the software system 10. The operating system 18 may or may not provide threads support. In addition, any threads

Attorney Docket No.: 10981455

support that may be provided by the operating system 18 may differ in structure and function substantially from the structure and function of threads support defined by the standardized interfaces of the virtual
5   machine 12.

The two tier structure for thread support according to the present techniques is embodied in a threads interface layer (TIL) 14 in the virtual
10   machine 12 and a native threads interface layer (NTIL) 16. The TIL 14 provides a standard threads interface for the virtual machine 12 which does not depend on the underlying platform for the software system 10 while the NTIL 16 is adapted to the
15   particulars of the underlying platform including the operating system 18 which underlies the virtual machine 12.

The two tier structure provided by the TIL 14
20   and the NTIL 16 enhances the portability of the virtual machine 12 to differing platforms having differing operating systems. For example, the virtual machine 12 may be ported to a different platform in which the TIL 14 provides the standard
25   threads interface to software tasks that execute under the virtual machine 12 and in which an NTIL implemented for the different platform is used to adapt the TIL 14 to the particulars of the operating system of the different platform.
30

In addition, the two tier arrangement provided by the TIL 14 and the NTIL 16 simplifies adaptation of the virtual machine 12 to changes in threads

support in the underlying functionality of the
operating system 18. For example, the NTIL 16 may be
modified or a new NTIL provided to adapt to the
changes in the operating system 18. The TIL 14
5   remains unchanged and continues to provide the
standard threads interface to the application program
20.

The two tier arrangement for thread support
10   according to the present techniques is illustrated
with respect to a set of threads 30-32 which are
associated with the application program 20. The
threads 30-32 may be defined as relatively small
processes that share an address space and that are
15   executed by the virtual machine 12 in parallel. The
parallel execution may involve context switching
among the threads 30-32 if the hardware platform for
the software system 10 provides a single processor
architecture. Alternatively, the threads 30-32 may
20   execute concurrently if the hardware platform for the
software system 10 provides a multi-processor
architecture or a multi-thread processor in hardware.

The functions performed by the threads 30-32
25   generally depend on the nature of the application
program 20. For example, if the application program
20 services a communication socket which receives
hypertext transfer protocol (HTTP) commands then each
thread 30-32 may have been created to handle a
30   different HTTP command received via the communication
socket.

able interface of Java application programming
interface (API). The following is a list of the
methods in the Java threads class in one embodiment.

```
run()
start()
stop()
suspend()
resume()
getThreadGroup()
getPriority()
setPriority()
isDaemon()
setDaemon()
countStackFrames()
join()
interrupt()
isInterrupted()
interrupted()
currentThread()
activeCount()
enumerate()
dumpStack()
yield()
sleep()
destroy()
```

(line numbers 5, 10, 15, 20, 25)

The methods in the TIL 14 have analogous methods
in the NTIL 16 which are adapted to the underlying
platform of the operating system 10. For example,
the TIL 14 provides methods for spawning the threads
30-32 in terms of the virtual machine 12. The NTIL
16 includes analogous methods for spawning user level
threads for the threads 30-32 in terms of the
underlying platform of the software system 10.
Similarly, the TIL 14 includes methods for suspending
and resuming Java threads in terms of the virtual
machine 12 and the NTIL 16 includes analogous methods
of suspending and resuming the corresponding user
level threads in terms of the platform that underlies
the software system 10.

In the example embodiment, the spawn
functionality of the TIL 14 is provided by an

implementation of the start() method of the Java
threads class. The spawning of a Java thread by the
TIL 14 includes the allocation of the Java stack and
initialization of the virtual machine registers SP,
5   PC, FP, LVP and NPC for the Java thread. This
creates the Java virtual machine context for the Java
thread. The TIL 14 then uses analogous methods in
the NTIL 16 to spawn a user level thread in terms of
the underlying platform of the software system 10.
10   The TIL 14 associates the user level thread created
by the NTIL 16 to the Java thread.

The following illustrates the context of each
thread 30-32 in terms of the virtual machine 12 in
15   the example embodiment.

```
     JVM_byte    *pc;       > current program counter
     JVM_byte    *npc;      > next program counter
     int         sp;        > top of Java app stack
20   int         lvp;       > local variable pointer
     int         fp;        > frame pointer
     JVM_word    *stack;    > base of Java app stack
     char        *refstk;   > base of the reference stack
     jmp_buf     jbuf;      > jump buffer for external
25                          exceptions
```

In one embodiment, the thread support methods in
the NTIL 16 are as follows.
```
     Thread threadCreate(...)
30   int threadSuspend(Thread)
     int threadResume(Thread)
     int threadJoin(Thread)
     int threadYield()
     int threadStop(Thread)
35   int threadSetPrio(Thread)
     int threadGetPrio(Thread)
     int threadCurrent()
     int scheduler()
```

40   The NTIL 16 provides an API for the above-listed
methods to the TIL 14. The methods in the TIL 14 use
this API to send requests to the methods in the NTIL

Attorney Docket No.: 10981455

16 to perform the particular threads functions in
terms of the underlying platform of the software
system 10. The API to the NTIL 16 is independent of
the underlying platform for the software system 10

5   and is independent of the particular implementations
of the methods in the NTIL 16. The following sets
forth one possible embodiment of the methods in the
NTIL 16.

10   The following structure is used to define a user
level thread in the context of the underlying
platform for the software system 10. PTHREAD refers
to a user level thread in terms of the underlying
operating system 18.

15

```
typedef struct thread_t{
        Lock        lock;          /* Lock for this thread structure
                                   */
        pthread_t pthr_id;         /* The id of the corresp. PTHREAD
20                                 */
        Utf8Const *name;           /* The thread name */
        JVM_word  state;           /* The tread state */
        Object    *thrObj;         /* Object corresp. to this thread
                                   */
25      VmContext vm;              /* Ptr to Virt. Mach. Rntime Info
                                   */
        Thread_t  *nest;
} * Thread;
```

30   The NTIL 16 uses Thread as a pointer to a thread
structure for a user level thread. Thread is an
unsigned integer that is used as a thread identifier
by the NTIL 16. The NTIL 16 maintains a global
thread list (GTL) which is a linked list of all user

35   level threads. One global lock is used for adding or
deleting thread structures from the GTL. A thread
identifier for a currently running thread is
maintained in a global variable which changes every

time the scheduler() method chooses to schedule another
thread.

     In this example embodiment, the NTIL 16 performs
a context switch using a swapcontext() routine which is
provided in a C implementation in the operating
system 18. The swapcontext() routine swaps the
internal registers and stack associated with the
particular processor of the underlying platform of
the software system 10.

     Figure 2 shows a state diagram for the threads
30-32 in one embodiment. The valid states of the
threads 30-32 include a READY state, a SUSPENDED
state, a RUNNING state, and a DEAD state. The state
diagram shows the transitions among the states caused
by the thread support methods in the NTIL 16.

     The method threadCreate() creates a new user level
thread by allocating a thread structure and calling
threadSpawn().

```
Thread threadCreate(void* (*startRoutine)(void*), void *a rgs,
                 Object *thrObj, Utf8Const *name)
{
        Allocate a Thread structure.
        Grab the lock for the global thread list (GTL).
        Fill in the defaults into the new Thread structure.
        Attach the current structure to the GTL.
        Set Thread State to READY.
        threadSpawn(&pthr_id,startRoutine, arguments);
        Return the lock for the GTL.
        Return the pointer to the Thread Structure.
}
```

     The method threadSuspend() suspends the specified
thread.

```
int threadSuspend(Thread t)
```

Attorney Docket No.: 10981455

```
{
        Lock thread struct
        Set state of t to SUSPEND
        if (t is currentThread){
                Unlock
                scheduler()
        }
        else
                Unlock
}
```

The method threadResume() resumes the specified thread.

```
int threadResume(Thread t)
{
        if (t is currentThread)
                return
        Lock thread struct
        if (state of t is SUSPENDED)
                Set state of t to READY
        Unlock thread struct
}
```

The method threadJoin() waits for completion of the specified thread.

```
int threadJoin(Thread t)
{
        if (t is currentThread)
                return
check:
        Lock Thread Struct
        if (State of t is DEAD)
                Unlock and return

        Attach on the thread wait set
        Unlock
        threadSuspend(currentThread)
        toto check;
}


int threadJoinTimeOut(Thread t, long msec)
{
        if (t is currentThread)
                return
check:
        Lock Thread Struct
        if (State of t is DEAD)
                Unlock and return

        Attach on the thread wait set
        Unlock
```

Attorney Docket No.: 10981455

```
                Set timer to wake up after msec.
                threadSuspend(currentThread)
                goto check;
        }
5

        The method threadYield() yields the execution,
i.e. processor time, to some other thread.


        int threadYield()
10      {
                scheduler()
        }


        The method threadStop() is called to stop a
15 specified thread and to clean up some of the
structures associated with the specified thread.


        int threadStop(Thread t)
        {
20              Lock Thread Struct
                Set state of t to DEAD
                Resume all threads waiting on t.
                Unlock
                Lock the GTL
25              Remove t from the GTL
                Unlock GTL
                if (t == currentThread)
                        scheduler()
        }
30

        The method threadSetPrio() sets the priority of the
specified thread.


        int threadSetPrio(Thread Struct)
35      {
                Lock Thread Struct
                Set priority of t to prio, if valid
                Remove t and attach to appropriate place in list.
                Unlock
40      }


        The method threadGetPrio() returns the priority of
the specified thread.


45      int threadGetPrio(Thread Struct)
        {
                Lock Thread Struct
```

```
        prio = priority of t
        Unlock
        return prio;
}
```

The method threadCurrent() returns the identifier
of the currently running thread.

```
int threadCurrent()
{
        return currentThread;
}
```

The method scheduler() selects a thread for
execution.

```
int scheduler()
{
        Lock the GTL
        c= Select the highest priority READY thread from the
(list)
                queue.
        T = currentThread;
        if (c == t)
                return;
        if (c == NULL)
                Fatal Error : Dead Lock Condition
        Set state of c to RUNNING
        Put c in the state of list.
        set currentThread = c
        if (state of t is RUNNING)
                Set state of t to READY
        Attach t to appropriate place in list.
        Unlock GTL
        threadContextSwitch(t,c);
}
```

```
threadContextSwitch(Thread from,Thread to)
{
        switchVMachineContext(from,to)
        /*
          Make sure that out of all the pthreads corresponding to
          Java threads only one pthread is READY at any given
time.
          This ensures the actual scheduling of the required Java
          application thread.
        */
        pthread_continue(to.pthr_id);
        pthread_suspend(from.pthr_id);
}
```

```
threadSpawn(pthr_id, void* (*startRoutine)(void*), void
*arguments)
```

```
{
        Call pthread_create(&pthr_id,startRoutine,arguments)
}
```

5   In some embodiments, the native thread support
routines of the NTIL 16 take advantage of thread
support routines that exists in the operating system
18. For example, some UNIX operating systems and the
LINUX operating system include an implementation of
10   pthreads which may be used by the NTIL 16. This is
illustrated by the example embodiment in which the
operating system 18 includes the thread support
routines pthread_create() and pthread_continue() and
pthread_suspend() which are invoked by the routines
15   threadContextSwitch() and threadSpawn() of the NTIL 16.
The routine pthread_create() creates a new thread in the
context of a processor under which the operating
system 18 executes at a specified address. The
routine pthread_create() is used to start a new
20   execution context and it returns a thread identifier
pthr_id for the new native thread. The routine
pthread_suspend() takes a thread identifier pthr_id as an
argument and is used to suspend the specified native
thread. The routine pthread_continue() takes a thread
25   identifier pthr_id as an argument and is used to
resume the specified native thread.

       In other embodiments of the software system 10,
the operating system 18 provides thread support
30   routines that perform equivalent functions to one or
more of the native thread support routines of the
NTIL 16. In such embodiments, the equivalent native
thread support routines of the NTIL 16 perform a call
to the appropriate thread support routine in the
35   operating system 18. For example, if the operating

Attorney Docket No.: 10981455

system 18 includes a thread scheduler routine that performs the equivalent function of the scheduler() routine in the NTIL 16, then the scheduler() routine in the NTIL 16 calls the thread scheduler routine of the operating system 18. Other native thread support routines may be fully coded in the NTIL 16 if equivalents are not available in the operating system 18.

In other embodiments, one or more of the native thread support routines may be fully coded in the NTIL 16 whether or not equivalents are provided in the operating system 18. This insulates the NTIL 16 from changes to thread support in the underlying operating system 18.

Figure 3 shows a method for providing threads support for a virtual machine in a software system. The method steps shown may be used to adapt the virtual machine to a new platform or to adapt the virtual machine to changes in the underlying platform such as changes to the underlying operating system and/or hardware.

At step 100, a threads interface layer is implemented in the virtual machine. The threads interface layer includes a set of methods that provide thread support according to a standard threads interface associated with the virtual machine. The methods in the threads interface layer maintain a set of context information for each of a set of threads in the software system in terms of the virtual machine.

Attorney Docket No.: 10981455

At step 102, a native threads interface layer is implemented. The native threads interface layer includes a set of methods that adapt the methods of the threads interface layer to the platform which

5 underlies the software system. The methods in the native threads interface layer maintain a set of context information for each of a set of threads in the software system in terms of the platform. Step 102 may be used to adapt the virtual machine to

10 changes in the underlying platform for a given virtual machine.

The foregoing detailed description of the present invention is provided for the purposes of

15 illustration and is not intended to be exhaustive or to limit the invention to the precise embodiment disclosed. Accordingly, the scope of the present invention is defined by the appended claims.

Attorney Docket No.: 10981455